Breadcrumbs Reasoning: Memory-Efficient Reasoning with Compression Beacons

Giovanni Monea[♦] **Yair Feldman**[♦] **Shankar Padmanabhan**[♦]
Kianté Brantley[♥] **Yoav Artzi**[♦]
[♦]Cornell University [♥]Harvard University
{giovanni,yairf,shankarp,yoav}@cs.cornell.edu kdbrantley@g.harvard.edu

Abstract

The scalability of large language models for long-context reasoning is severely constrained by the linear growth of their Transformer key-value cache, which incurs significant memory and computational costs. We posit that as a model generates reasoning tokens, the informational value of past generated tokens diminishes, creating an opportunity for compression. In this work, we propose to periodically compress the generation KV cache with a learned, special-purpose token and evict compressed entries. We train the model to perform this compression via a modified joint distillation and reinforcement learning (RL) framework. Our training method minimizes overhead over the conventional RL process, as it leverages RL outputs for distillation. Empirically, our method achieves a superior memory–accuracy Pareto frontier compared to both the model without cache compression and training-free compression techniques.

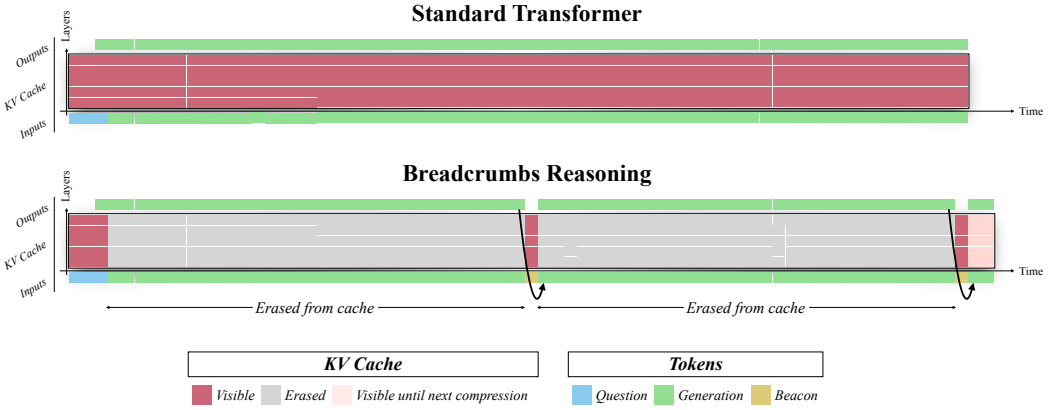

Figure 1: **Breadcrumbs Reasoning,** with a compression ratio $c = 16$. To save memory during inference, a window of c tokens is periodically compressed into a single beacon token. The original KV cache entries for the window are then evicted, leaving only a compact 'breadcrumb' that summarizes the preceding reasoning steps.

1 Introduction

Reasoning through token generation allows large language models (LLMs) to solve arbitrarily complex problems with a fixed depth architecture (Merrill & Sabharwal, 2023), by scaling the compute invested through the generation of more tokens (i.e., test-time scaling) (Snell et al., 2024). This

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

practice carries high computational costs, because of the self-attention design of Transformers (Bahdanau et al., 2014; Vaswani et al., 2017; Keles et al., 2023). Not only it relies on simply generating many more tokens, but later tokens require computation over the representations of all previous tokens, incurring higher time complexity and necessitating increasing memory costs.

However, not all past representations are equally important. For example, the details of a previously explored attempt at a solution are likely not critical, as long as the model retains some signal that advises it to avoid exploring this same failed path again. We propose to jointly learn to reason, compress, and discard previously computed representations along reasoning chains.

The key to our approach is to substitute previously computed key-value (KV) cached representations with significantly more compact representations, inspired by how activation beacons are used for long-context compression (Zhang et al., 2025). We train these representations to contain the information from past tokens that is necessary for continuing the reasoning process to solve the task the model is given, allowing us to evict from the KV cache most previously computed representations. The challenge is to combine the training of these beacons into the reinforcement learning (RL) (Sutton & Barto, 2018) process that makes length-based reasoning possible, a fundamentally different process than the pre-training that enables conventional long-form generation. We design a joint RL-distillation approach, where we train the original non-compression policy using the conventional RL process with a verifier for reward computation, and concurrently distill it into a policy that jointly compresses and reasons.

We evaluate our method, Breadcrumbs Reasoning (Figure 1), on the Qwen2.5-1.5B and Phi-4-Mini models across three challenging reasoning benchmarks: Countdown (Gandhi et al., 2024), LinSys, and StarGraph (Bachmann & Nagarajan, 2024). We compare against a strong, uncompressed teacher policy trained with RL, as well as two training-free cache eviction baselines, TOVA (Oren et al., 2024) and StreamingLLM (Xiao et al., 2023). Our experiments reveal several key findings. Demonstrating a clear Pareto improvement, Breadcrumbs Reasoning enables effective test-time scaling by generating longer reasoning chains to match or exceed the teacher's accuracy within a fixed memory budget, while still retaining 65.1–89.8% of the original performance when using 2–32x less memory at a fixed generation length. In contrast, the training-free baselines consistently underperform, stressing the necessity of a learned compression scheme for complex reasoning. We also validate our training strategy, showing that our joint RL-distillation approach matches or outperforms a more complex two-stage training pipeline, confirming its efficiency. Our code is available at `https://github.com/lil-lab/breadcrumbs-reasoning`.

2 Related Work and Background

Generation in Transformers-based LLMs requires reasoning over and storing a key-value (KV) cache. This entails high memory (i.e., space) and time costs, which increase as the context (i.e., the number of previous tokens) increases. Therefore long-form generation costs suffer not only from the fundamental need to generate more tokens via more steps, but also from the increasing cost of each such step. KV compression is a solution avenue that is receiving significant research attention (LI et al., 2025).

An important thread within this compression literature is training models to perform KV cache compression. Nawrot et al. (2024) train models to compute importance scores, which are then used to store averaged KV cache entries instead of the original entries. Other methods train the Transformer-based LLMs themselves to summarize past KV entries (Chevalier et al., 2023; Zhang et al., 2025), so more compact representations can be retained. Our approach is inspired by the activation beacons method (Zhang et al., 2025), but with significant simplifications and adaptation for reasoning. We do away with the chunk and sub-chunk distinction, and eliminate the addition of specialized attention mechanisms. Rather, we adopt a flat segmentation into blocks, use the standard Transformer attention mechanism, and remove KV cache entries every time a beacon is processed (e.g., ex-ante). These modifications do not only simplify the implementation, but also allow for immediate eviction of cache entries, instead of a delayed one. More broadly, a drawback of these learned methods is that they require fine-tuning on a considerable amount of general-purpose pre-training data. We design a joint reasoning-compression training approach, which adds minimal overhead over the existing reasoning training processes.

Algorithm 1 Breadcrumbs Reasoning

Input: Transformer-based policy π_{BR}, beacon token b, prompt tokens $\bar{q}$, stop token s, compression ratio c. Let $\mathrm{KV}_{\pi_{\mathrm{BR}}}$ be the persistent KV cache of the policy model π_{BR}.
Output: $\bar{x}$

Initialize: Encode $\bar{q}$ through π
for $i = 0, 1, 2, \ldots$ **do**
  $x_i \sim \pi_{\mathrm{BR}}(\cdot|\bar{q}, \bar{x})$ ▷ *Sample the next token.* $\mathrm{KV}_{\pi_{\mathrm{BR}}}$ *is updated internally.*
  $\bar{x} \leftarrow \bar{x} + x_i$ ▷ *Concatenate the sampled token to the end of the output.*
  if $x_i = s$ **then** ▷ *Check for the generation stopping token.*
    break
  if $i > 0$ **and** $i \bmod c = 0$ **then**
    Encode b through $\pi_{\mathrm{BR}}(\cdot|\bar{q}, \bar{x})$ ▷ *Updates* $\mathrm{KV}_{\pi_{\mathrm{BR}}}$ *with the entry for the compression token* b.
    $\mathrm{KV}_{\pi_{\mathrm{BR}}} \leftarrow \mathrm{KV}_{\pi_{\mathrm{BR}}}[:-c-1]$ ▷ *Drop the KV cache entries of the most recent* c *tokens.*
      $+\mathrm{KV}_{\pi_{\mathrm{BR}}}[-1]$ ▷ *But, keep the entry of the beacon* b.
return $\bar{x}$

An alternative to training-based methods are training-free methods that perform compression at generation time. They can be divided into two main categories. The first is that of sliding-window approaches, which limit the KV cache by only including a sliding window plus an additional subset of tokens. Particularly simple and effective is StreamingLLM (Xiao et al., 2023), which finds that including a few initial *sink* tokens in addition to the window recovers most of the uncompressed performance. The second type does not constrain window tokens to remain in the cache, but rather tries to select the empirically more important for attention computations, as in H2O (Zhang et al., 2023) or TOVA (Oren et al., 2024).

An orthogonal approach to reducing the KV cache size is reducing Chain-of-Thought reasoning length (Kang et al., 2025; Ma et al., 2025; Shen et al., 2025; Yan et al., 2025; Munkhbat et al., 2025; Xia et al., 2025). These methods primarily aim to directly shorten reasoning traces. This is distinct from our objective of dynamic KV cache compression, which focuses on extracting critical information to manage cache sizes effectively during the reasoning process itself. KV cache compression methods, like ours, could be applied in combination with those methods to achieve even higher levels of efficiency.

3 Methodology

Breadcrumbs Reasoning (BR) periodically computes compressed representations of KV cache entries and evicts them from the KV cache. We design a training process that adds relatively little overhead on top of the conventional reasoning RL process. The learned policy model effectively reasons through token generation and concurrently compresses KV cache representations.

3.1 Breadcrumbs Reasoning

Our compression scheme uses the Transformer architecture itself for both compression and reasoning. We generate tokens following the same procedure as a vanilla Transformer-based language model, but periodically compute compressed representations of past KV cache entries, and evict these entries from the cache. Algorithm 1 describes the process. We use a special token b to mark when the model should compute compressed representations of past tokens, and input this token every c tokens, where c is the target compression ratio. The KV cache entries for the token b form the compressed representation, and we drop the entries for previous c tokens. Immediately after the beacon b, we force the next input token to be the last sampled token before b was given as input. This is equal to continuing conventional generation. Figure 1 visualizes this process to illustrate the space savings. Roughly speaking, this process leaves a trace of "reasoning breadcrumbs" behind, instead of long, detailed, and eventually irrelevant information.

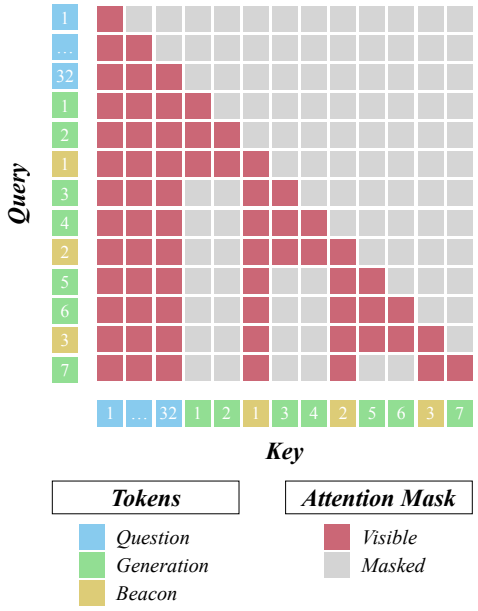

Figure 2: **Attention mask used to enforce compression during training.** Each token after the question can attend to the initial question tokens, all previous beacons, and earlier generation tokens within the same window, i.e., since the most recent preceding beacon. This encourages the model to compress relevant past context into the beacons to support future generations.

3.2 Joint RL-distillation Training

Typically, LLMs are trained to solve reasoning tasks through RL (Shao et al., 2024; DeepSeek-AI, 2025; Lambert et al., 2025). Applying this process as is to a breadcrumbs reasoning policy is technically possible, but is unlikely to lead to effective learning. This is because, before training, the model does not have compression ability, so incorporating the compression token and cache eviction will significantly damage its functionality, and it will observe no positive reward during RL.

We use a surrogate teacher policy π_{RL} that does not compress and is trained through RL to perform the reasoning task, and distill it into the breadcrumbs policy π_{BR}. By learning to imitate the surrogate policy π_{RL}, our target breadcrumbs policy π_{BR} simultaneously learns to compress and to perform the new reasoning task. This process relies on the trajectories sampled during RL, so no expensive sampling of trajectories beyond the conventional RL process is needed. This procedure minimizes the overhead over a standard two-steps procedure, where either (a) π_{RL} is completely trained before generating new data for π_{BR} to learn to compress, or (b) π_{BR} is trained to compress through extensive general data and only after that learns the new reasoning task.

Given a token trajectory $\bar{x}$ sampled from π_{RL}, the distillation loss is defined as the average token-level KL divergence between the uncompressed policy π_{RL} and the compressed policy π_{BR}:

$$
\begin{aligned}
L(\bar{x}) &= \frac{1}{|\bar{x}|} \sum_{i=1}^{|\bar{x}|} D_{KL}\left(\pi_{\mathrm{RL}}(x_i|\bar{x}_{<i}) \parallel \pi_{\mathrm{BR}}(x_i|\bar{x}_{<i})\right) \qquad (1)\\
&= \frac{1}{|\bar{x}|} \sum_{i=1}^{|\bar{x}|} \sum_{k=1}^{|V|} \pi_{\mathrm{RL}}(x_i = k|\bar{x}_{<i}) \log \frac{\pi_{\mathrm{RL}}(x_i = k|\bar{x}_{<i})}{\pi_{\mathrm{BR}}(x_i = k|\bar{x}_{<i})}
\end{aligned}
$$

The total loss is then computed as the average over a batch of B trajectories:

$$
L = \mathbb{E}_{\bar{x}\sim\pi_{\mathrm{RL}}}\left[L(\bar{x})\right] \approx \frac{1}{B} \sum_{b=1}^{B} L(\bar{x}_b)
$$

When training π_{BR}, we only compute the gradient of L with respect to the parameters of π_{BR}. The parameters of the teacher policy π_{RL} are kept frozen during this update step, as π_{RL} should not adapt to π_{BR}.

For parallel and efficient training, at training time π_{BR} does not execute the KV cache removal strategy (Section 3.1), but simulates it by masking compressed tokens. Figure 2 visualizes the attention mask that results from the training masking pattern.

4 Experimental Setup

Tasks We use three reasoning tasks that the initial models solve with only a very low success rate:

Countdown (Gandhi et al., 2024): the task input is a tuple of numbers, and the goal is to create a sequence of arithmetic operations using a subset of the numbers to equal a target number. While this task requires the model to avoid repeating previous attempts, or it can enter an endless loop, the individual guesses are largely independent of one another.

LinSys: each problem consists of a linear equation system with a unique integer solution. The coefficients and variables are randomly generated. In contrast to Countdown, which emphasizes educated trial and error, this task more closely reflects structured reasoning: solving for one variable often enables solving for the next, resulting in a multi-step deductive process.

StarGraph (Bachmann & Nagarajan, 2024): the model is given a list of directed edges of a star graph (i.e., a graph with multiple branches of a fixed length all expanding from a central node) and a target end node. The model must output the full sequence of edges to get to the target node from the central node. This is a task auto-regressive models such as Transformers naturally struggle with, as observed by Bachmann & Nagarajan (2024) and Hu et al. (2025).

Model and Training Details We utilize two models for our experiments: Qwen2.5-1.5B-Instruct (Team, 2025) and Phi-4-mini-instruct (Microsoft, 2025). For Qwen2.5-1.5B-Instruct, we train for 1000 steps for all tasks, while for Phi-4-mini-instruct, we train for 200 steps for StarGraph and LinSys and 1000 steps for Countdown. We use a batch size of 256 for both models. We use PPO (Schulman et al., 2017) as the RL algorithm for π_{RL}. We experiment with compression ratios c of 2, 4, 8, 16, and 32 for breadcrumbs reasoning. The reward for an incorrect or not present answer is 0.0, a correct format of the final answer but an incorrect value gets a 0.1 reward, and a correct response gets a 1.0 reward.

Baselines We compare our approach against two primary training-free baselines applied to π_{RL}: StreamingLLM (Xiao et al., 2023) and TOVA (Oren et al., 2024). Similar to our joint training setup, they also do not require more data than what π_{RL} was trained on, and they also delete entries from the KV cache.[1] For a fair comparison, we adapt these methods to use an increasingly large KV cache size during generation, matching the memory footprint of our approach. This is to avoid potentially penalizing them with a smaller static cache size. In particular, given a compression ratio c, we allow the cache to grow by 1 every c generated tokens. We set the sliding window of StreamingLLM to c to match the budget given to our approach. We also set the sink size of StreamingLLM to the entire question. For TOVA, we set the initial size of the KV cache to the number of question tokens plus c, so that the first compression would only happen after at least c tokens, similar to our method. For both methods, we test the same c as for our policies. We also study the effect of two-step training, by first training the RL policy π_{RL} and only later generating data for distillation. In both cases, we use the same number of data samples and training steps (256 samples per step, for 1k steps).

Evaluation We generate evaluation answers for 256 held-out test examples for each task. In all settings, we fix the maximum KV cache size to 1,000 entries (i.e., tokens), which is also the maximum response length permitted during training. Generation is interrupted if this limit is reached.

5 Results

We compare the Breadcrumbs Reasoning (BR) policy to an uncompressed RL policy and to the baselines in two different configurations. We first compare results with a set maximum cache size (Table 1 and Figure 3). We also compare with a fixed maximum number of generation tokens (Table 2), the same used during training of π_{RL}.

Test-Time Scaling with Breadcrumbs Reasoning Table 1 shows accuracy performance with a fixed maximum cache budget of 1,000 steps, as well as the accuracy area under the curve (AUAC) with varying the maximum cache size up to 1,000. Figure 3 shows the curves. Across most settings,

[1] We omit comparing to methods such as QUEST (Tang et al., 2024), because they are not focused on compression, but rather smart management of the GPU memory, an orthogonal approach to compression.

		Countdown				LinSys				StarGraph			
		Qwen		Phi		Qwen		Phi		Qwen		Phi	
Method	c	Acc_c	AUAC	Acc_c	AUAC	Acc_c	AUAC	Acc_c	AUAC	Acc_c	AUAC	Acc_c	AUAC
Teacher	-	0.598	0.377	0.633	0.433	0.918	0.276	0.898	0.142	0.902	0.513	0.848	0.498
Breadcrumbs	2	0.605	0.465	0.609	0.506	0.730	0.451	0.652	0.373	0.957	0.726	0.812	0.636
	4	0.613	0.533	0.625	0.557	0.656	0.532	0.539	0.421	0.969	0.845	0.832	0.739
	8	0.613	0.555	0.613	0.581	0.410	0.368	0.363	0.322	0.973	0.906	0.836	0.786
	16	0.574	0.539	0.625	0.604	0.367	0.347	0.219	0.205	0.957	0.918	0.812	0.785
	32	0.535	0.513	0.613	0.597	0.297	0.286	0.195	0.187	0.957	0.926	0.816	0.792
TOVA	2	0.574	0.447	0.230	0.196	0.000	0.000	0.000	0.000	0.664	0.526	0.801	0.621
	4	0.289	0.272	0.066	0.064	0.000	0.000	0.000	0.000	0.457	0.417	0.562	0.504
	8	0.172	0.167	0.051	0.050	0.000	0.000	0.000	0.000	0.445	0.424	0.457	0.433
	16	0.188	0.183	0.047	0.046	0.000	0.000	0.000	0.000	0.430	0.414	0.453	0.436
	32	0.207	0.199	0.098	0.094	0.000	0.000	0.000	0.000	0.441	0.423	0.395	0.378
StreamingLLM	2	0.012	0.007	0.016	0.014	0.000	0.000	0.000	0.000	0.055	0.038	0.180	0.137
	4	0.023	0.015	0.031	0.018	0.000	0.000	0.000	0.000	0.055	0.032	0.020	0.017
	8	0.027	0.024	0.109	0.060	0.000	0.000	0.000	0.000	0.031	0.023	0.031	0.029
	16	0.051	0.049	0.156	0.150	0.000	0.000	0.000	0.000	0.047	0.041	0.098	0.089
	32	0.094	0.090	0.312	0.300	0.000	0.000	0.000	0.000	0.117	0.105	0.102	0.097

Table 1: **Model performance on long-context reasoning tasks for a fixed generation cache size**. We report accuracy given a maximum cache size of 1,000 entries (Acc_c) and Area Under the Accuracy Curve (AUAC). c denotes the compression ratio.

BR recovers most of the performance of the teacher at a much lower memory cache budget. Except for LinSys, which remains challenging, all compression ratios outperform the teacher for most of the budgets. This is because the compressed models are able to effectively accommodate more reasoning steps within the same cache budget, allowing for more aggressive test-time compute scaling. On Countdown, both BR models outperform even the teacher at maximum KV cache budget.

We measure the trade-off between accuracy and KV cache consumption by measuring the area under the accuracy-cache size curve. This metric integrates accuracy over the full range of cache sizes, and thus captures a model's overall robustness to a variety of cache constraints. A larger AUAC indicates that a method maintains high accuracy across a wider range of cache budgets, rather than only performing well with high memory usage. We can observe that BR outperforms even the teacher policy by 3.6–92.8% for Qwen and 16.9—196.5% for Phi across the different ratios and tasks. For high compression ratios, the number of generated tokens is much higher than during training (e.g., 32,000 tokens vs 1,000 for 32x compression ratio). This is evidence that compression generalizes to much longer sequences than those observed during training.

We visualize this effect in Figure 4. In particular, Qwen and Phi improve by up to 14.5% and 5.1% across tasks after 1,000 generation tokens. The only exception is LinSys, where more generation tokens do not seem to help (as can be seen from Figure 3 too).

Beyond matching teacher accuracy at full context, BR is far more efficient with cache usage. In Countdown and StarGraph, Figure 3 shows that BR nearly reaches or even exceeds the teacher's peak accuracy with fewer than 250 tokens in cache - less than a quarter of the teacher's required 1,000 tokens. For instance, in Phi–Countdown, BR surpasses 0.60 accuracy by cache size 100, while the teacher requires more than 800 tokens to reach the same level. A similar pattern holds in StarGraph, where BR consistently reaches $>$0.80 accuracy with only a few hundred tokens, whereas the teacher climbs much more gradually. The only clear exception is LinSys, where BR improves more slowly and is not able to completely close the gap to the teacher.

Breadcrumbs Reasoning Retains Most of Uncompressed Performance We compare the BR policies across different compression ratios to the π_{RL} that we use as the distillation source. We maintain the same source policy for all our experiments to make this comparison valid. Table 2 reports the results for a fixed maximum generation length (i.e., time steps) of 1,000.

While on Countdown and StarGraph all compression ratios work well, performance on LinSys varies greatly across compression ratios. As expected, higher compression ratios lead to worse performance. For LinSys, BR lags behind the teacher for both models. We speculate this is due to differences in the reasoning challenge compared to Countdown and StarGraph. Overall, given a fixed

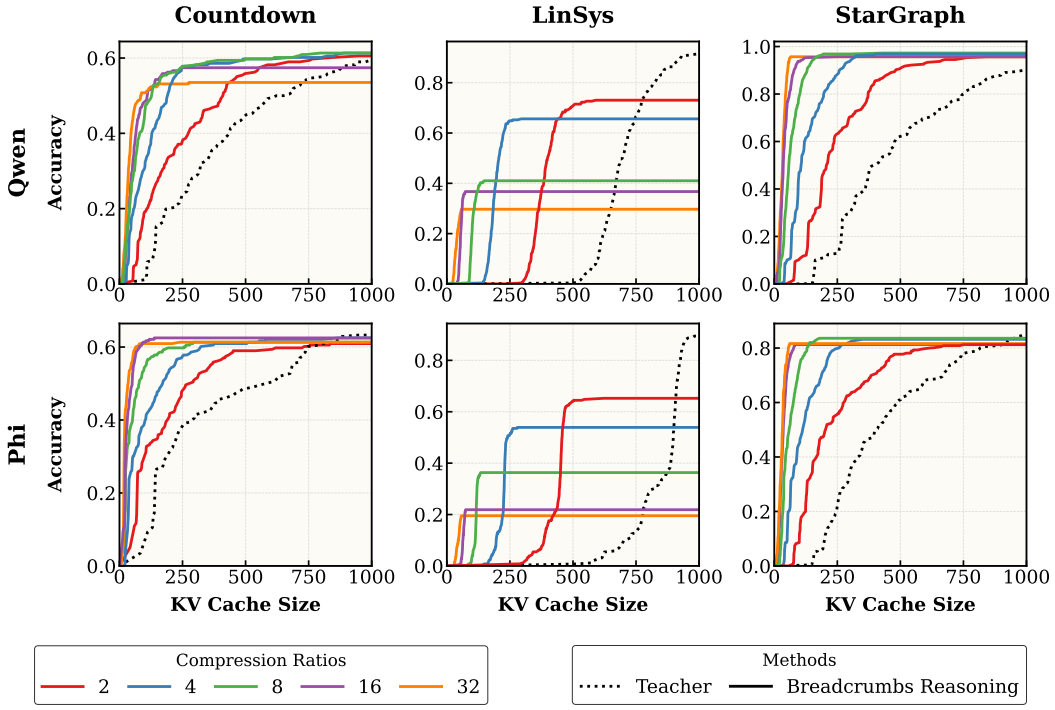

Figure 3: **Accuracy vs. KV Cache Size** Breadcrumbs Reasoning retains most of the teacher's performance while using significantly fewer KV cache entries, even outperforming the teacher when setting a fixed maximum KV cache size in Countdown and StarGraph.

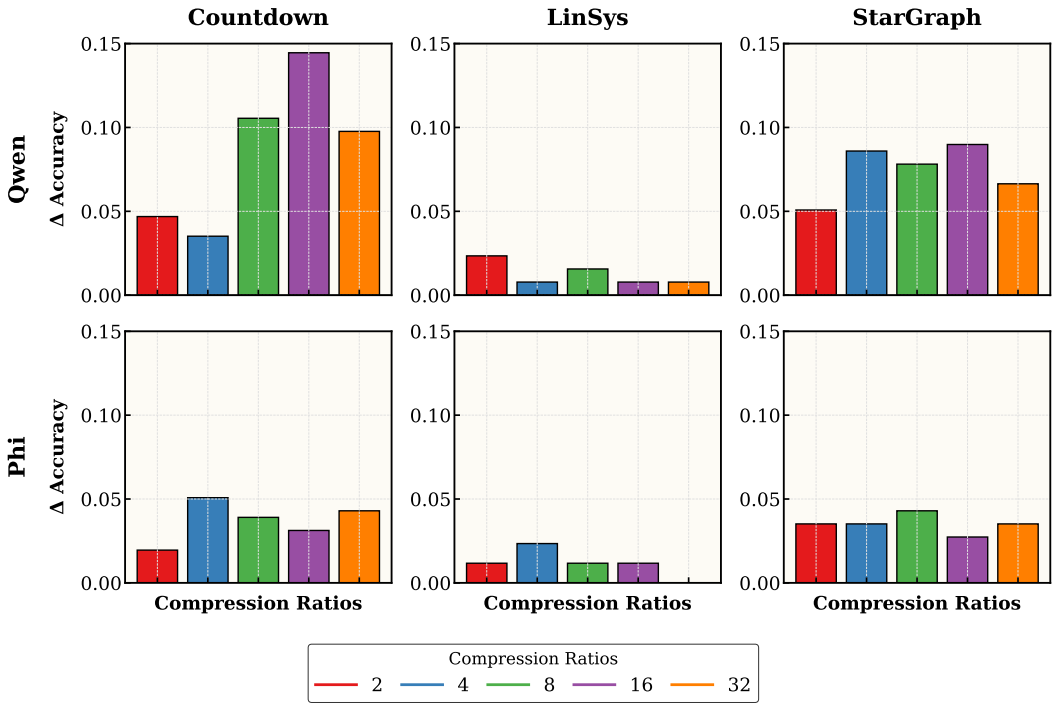

Figure 4: **Performance increase with extended generation.** Breadcrumbs Reasoning improves up to 14.5% with Qwen and 5.1% with Phi beyond the 1,000 token training length.

		COUNTDOWN		LINSYS		STARGRAPH		AVERAGE	
METHOD	c	QWEN	PHI	QWEN	PHI	QWEN	PHI	QWEN	PHI
TEACHER	-	0.598	0.633	0.918	0.895	0.902	0.848	0.806	0.793
BREADCRUMBS	2	0.559	0.590	0.707	0.641	0.906	0.777	0.724	0.669
	4	0.578	0.574	0.648	0.516	0.883	0.797	0.703	0.629
	8	0.508	0.574	0.395	0.352	0.895	0.793	0.599	0.573
	16	0.430	0.594	0.359	0.207	0.867	0.785	0.552	0.529
	32	0.438	0.570	0.289	0.195	0.891	0.781	0.539	0.516
TOVA	2	0.535	0.219	0.000	0.000	0.648	0.773	0.395	0.331
	4	0.289	0.066	0.000	0.000	0.457	0.559	0.249	0.208
	8	0.172	0.051	0.000	0.000	0.441	0.453	0.204	0.168
	16	0.188	0.047	0.000	0.000	0.430	0.453	0.206	0.167
	32	0.207	0.098	0.000	0.000	0.441	0.395	0.216	0.164
STREAMINGLLM	2	0.008	0.016	0.000	0.000	0.043	0.156	0.017	0.057
	4	0.012	0.012	0.000	0.000	0.016	0.016	0.009	0.009
	8	0.012	0.039	0.000	0.000	0.012	0.031	0.008	0.023
	16	0.051	0.137	0.000	0.000	0.012	0.078	0.021	0.072
	32	0.094	0.297	0.000	0.000	0.047	0.102	0.047	0.133

Table 2: **Model accuracy on long-context reasoning tasks for a fixed generation length**. The metric shown is accuracy at a sequence length of $L = 1{,}000$. c denotes the compression ratio. In particular, KV cache compression methods use only $\frac{1}{c}$ of memory compared to the teacher to achieve the reported accuracy.

response length, BR preserves performance across tasks by 67.1–94.0% for Qwen and 65.1–84.5% for Phi while using only 2–32x fewer KV cache entries at generation time.

Training-Free Methods Underperform The training-free methods TOVA and StreamingLLM consistently underperform across tasks. On Countdown, their performance drops dramatically with higher compression (e.g., TOVA falls from 0.574 at 2× to 0.172 at 8× for Qwen; StreamingLLM remains below 0.32 across all settings). On StarGraph, the gap between the baselines and out approach is even more severe, with StreamingLLM dropping below 0.1 accuracy in nearly every configuration. In LinSys, both baselines fail to reach even a single correct output. These results highlight the limitation of training-free cache eviction methods: for tasks requiring long coherent reasoning chains, simply truncating past tokens eliminates essential intermediate steps, from which the model is unable to recover.

Joint RL-Distillation Training Matches or Outperforms a Two-Steps Training We compare two strategies for distilling from π_{RL} in Figure 5. In our joint RL-distillation training, BR is distilled online from the rollouts of the teacher π_{RL} as it learns; in late distillation, compression policies are trained only on trajectories sampled from a final checkpoint of π_{RL}. BR achieves equivalent or superior performance in 26 of the 30 configurations tested, and very close performance on the other four. This demonstrates that it effectively piggybacks on the same data used to train π_{RL}. This eliminates the need for additional distillation data, minimizes training overhead, and does not impose the need to decide a priori a number of training steps for π_{RL}. We hypothesize that the superior performance may derive from the distributional shift of π_{RL} during RL training.

6 Discussion

We present a training-based approach to compress reasoning chains: Breadcrumbs Reasoning. Our empirical results demonstrate that indeed there is significant room for compression in reasoning chains, and not all information or tokens are equally important for downstream reasoning and even task completion. Our approach shows effective compression while retaining much of the reasoning performance. In contrast, training-free methods are significantly less effective.

We propose a joint RL–distillation training scheme, which provides an efficient way to teach a model to reason while also learning to compress. Training is necessary in any case to develop a

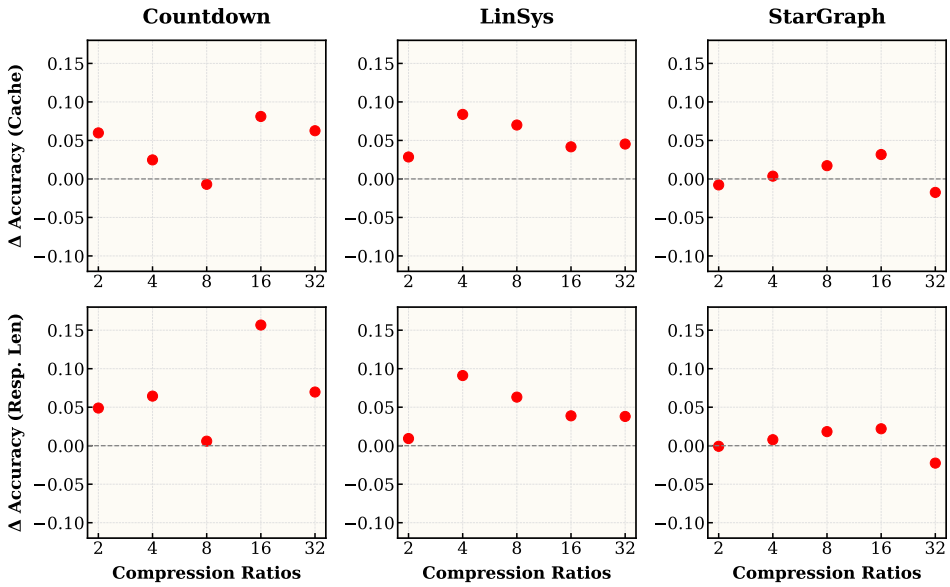

Figure 5: **Joint RL-distillation vs. Two-step Training on Qwen.** We compare our joint approach to the standard two-step process, where compression policies are trained on trajectories from a final π_{RL} checkpoint. Each point shows the accuracy difference (Joint − Two-step) at a given compression ratio. The top row fixes cache size; the bottom fixes response length. Positive values favor joint training, which outperforms or matches in 26/30 settings, showing BR can learn compression online during RL without separate distillation data.

policy that can successfully solve a reasoning task. Our approach integrates this requirement into a more effective procedure.

Although at a fixed generation length our method shows a small performance loss, we demonstrate that when generating many more tokens under the same memory budget, performance often surpasses that of a non-compressing teacher. This is enabled by effective test-time scaling: when matching the KV-cache size, we are able to generate significantly more tokens. To some extent, these results show that we trade memory for time. Such trade-offs are common in efficiency-oriented methods — for example, speculative decoding (Leviathan et al., 2023; Chen et al., 2023) reduces latency but requires more memory, since multiple models must be loaded simultaneously.

There are several directions for future work, entailed from several areas where our approach can be improved. Foremost, our compression policy is not dynamic — one cannot adjust its compression rate as desired. It is advantageous to have a single model that supports different compression rates, and therefore can be used flexibly in different settings. This is an important direction for future work, and one that is relatively understudied in the compression literature. Our work also charts the direction for future work to improve test-time scaling and compression tradeoffs. Ideally, one can compress aggressively without needing to increase the number of reasoning steps. Our work exposes this tradeoff in reasoning models, and outlines the methodology to analyze it. Finally, as the space of accessible benchmarking scenarios in reasoning research develops, it is important to understand the behavior of our approach across a broader set of domains.

References

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1547–1574. PMLR, 2024. URL `https://proceedings.mlr.press/v235/bachmann24a.html`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `https://api.semanticscholar.org/CorpusID:11212020`.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language, 2024. URL https://arxiv.org/abs/2404.03683.

Edward S. Hu, Kwangjun Ahn, Qinghua Liu, Haoran Xu, Manan Tomar, Ada Langford, Dinesh Jayaraman, Alex Lamb, and John Langford. The belief state transformer. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ThRMTCgpvo.

Simon Jegou, Maximilian Jeblick, and David Austin. kvpress, November 2024. URL https://github.com/NVIDIA/kvpress.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24312–24320, 2025.

Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *Proceedings of the 34th International Conference on Algorithmic Learning Theory*, volume 201 of *Proceedings of Machine Learning Research*, pp. 1–23. PMLR, 2023. URL https://proceedings.mlr.press/v201/duman-keles23a.html.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19298. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Haoyang LI, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole HU, Wei Dong, Li Qing, and Lei Chen. A survey on large language model acceleration based on KV cache management. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=z3JZzu9EA3.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. Cot-valve: Length-compressible chain-of-thought tuning. *arXiv preprint arXiv:2502.09601*, 2025.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought, 2023.

Microsoft. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras, 2025.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*, 2025.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, and Shiguo Lian. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *arXiv preprint arXiv:2503.04472*, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024.

Qwen Team. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models. *arXiv preprint arXiv:2503.06692*, 2025.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=1eQT9OzfNQ.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

A Implementation Details

We use VeRL (Sheng et al., 2024) as the backbone for our training code. We adapt kvpress (Jegou et al., 2024), a recent library for Transformers KV cache compression for inference (e.g., for evaluation). We decouple RL and distillation during our experiments, so that the teacher data is the same for all BR policies and results can be compared more fairly. Otherwise, different experimental results may depend on the different quality of π_{RL} training, rather than on the methods being tested. We save all π_{RL} sampled trajectories to decouple training, and the top-100 logits for each time step, because saving all logits is extremely storage-intensive. In all training setups, this leads to consistently distilling more than 90% of the probability mass for each token, although it frequently reaches even higher levels throughout training.

B Task Details

Countdown: The task is to combine 3–4 numbers using arithmetic operations to reach a target value. All target values are less than or equal to 100.

Countdown Example Prompt

Using the numbers `25, 10, 7, 3`, create an equation that equals `96`. You can use basic arithmetic operations (`+`, `-`, `*`, `/`) and each number can only be used once. Make sure to solve it by thinking step by step. Return the final answer in `<answer> </answer>` tags, for example `<answer> (1 + 2) / 3 </answer>`.

StarGraph: We use star graphs (Bachmann & Nagarajan, 2024) with branches of length 5 and up to 25 branches per graph. When creating the dataset, the number of branches for each graph is sampled uniformly from the range $[2, 25]$. We empirically found that this variable complexity helps the reinforcement learning policy, π_{RL}, by allowing it to gradually learn to solve more complex graphs.

StarGraph Example Prompt

You are given a star graph with the following nodes: `34, 72,` [...]`, 304`.

The graph has the following directed edges:
`34 -> 72`
[...]

Find the path from the center node `34` to the target node `304`.

Think step by step about the graph structure and trace the path from the center node to the target node.
Return your answer as a list of nodes representing the path from center to target.
Return the final answer in `<answer> </answer>` tags, for example `<answer>[1, 3, 7, 12]</answer>`.

LinSys: We generate systems of linear equations that have a single, unique solution. To ensure an appropriate level of difficulty for each model family, we used two distinct configurations:

- For Qwen models: We generated systems of 4 equations in 4 variables. Each equation has at most two non-zero coefficients, and the maximum absolute value for any coefficient is 20.
- For Phi models: We found the 4x4 configuration to be too simple (over 40% accuracy before training). We therefore created a more challenging setup: systems of 3 equations in 3 variables, with no restrictions on the number of non-zero coefficients and a maximum absolute coefficient value of 20.

PPO Hyperparameter	Value
Actor Learning rate	1×10^{-6}
Critic Learning rate	1×10^{-5}
Clip ratio (ϵ)	0.2
Number of epochs	1
Mini-batch size	256
Discount factor (γ)	1.0
GAE parameter (λ)	1.0
Entropy coefficient	0.001
Value loss coefficient	0.5
Clip range value	0.5
Max gradient norm	1.0

AdamW Hyperparameter	Value
Weight decay	0.01
β_1	0.9
β_2	0.999
Epsilon (ϵ)	1×10^{-8}

Table 3: Hyperparameters for PPO (left) and AdamW (right) for π_{RL}.

Hyperparameter	Value
Weight decay	0.01
β_1	0.9
β_2	0.999
Epsilon (ϵ)	1×10^{-8}

Table 4: Hyperparameters for AdamW optimizer for π_{BR}.

LinSys Example Prompt

Solve the following system of linear equations:
`2*x1 - x2 + 3*x3 + 4*x4 = 10`
`-x1 + 4*x2 - 2*x3 + x4 = 5`
`3*x1 + x2 + x3 - x4 = 12`
`x1 + 2*x2 + 5*x3 + 2*x4 = 20`
Find the values for `x1, x2, ..., x4`. Make sure to solve it by thinking step by step, and do not assume access to any external tools.
Return the final answer as a list of numbers in `<answer> </answer>` tags, for example `<answer>[1, -2, 3, 7]</answer>`.

C Training Details

C.1 π_{RL} Training

Table 3 provides the hyperparameters for PPO Schulman et al. (2017) and AdamW (Loshchilov & Hutter, 2019). The critic has the same architecture and initial weights as π_{RL}.

C.2 π_{BR} Training

Table 4 provides the hyperparameters for the AdamW optimizer. We use a learning rate of 5×10^{-6} for all tasks and models, except for the Countdown-Phi configuration, where we use a higher learning rate of 5×10^{-5}. This choice is based on empirical observations.